Lab 0: MPLAB X – Integrated Development Environment

Report

**Owen Blair  
9/7/2021**

# Introduction

The purpose of this lab is to be able to become familiar with the important aspects and tools in MPLAB that will be used while developing projects for the PIC32MX microcontroller. MPLAB will be the interface between the development platform where C code will be written and the compiler where the code will be compiled and loaded onto the PIC32MX. Some of the main features that will be used is the syntax checker, debug mode, link/load processed, breakpoints, and cycle counter.

# Implementation

Software implementation went as planned. This was easy because all that was needed was to define where the header and source files were and then compile the source code so it could be loaded into the microcontroller. No new code was written for this lab.

The important things in this lab are the no\_swap() and the swap() functions. In the while(1) loop the no\_swap() function does not use the & for a pointer and is unable to modify the a and b variables because of this. This can be seen in Listing 1 below.

## **Listing 1**

while(1) /\* Infinite application loop \*/

{

no\_swap(a, b); /\* Call swapping functions \*/

c = swap(&a, &b);

a = c - d; /\* Math statement \*/

//d=a-c; //LAB QUESTION

}

Within the swap() function there are several important things to note that makes the swap() function operate differently than the no\_swap() function. The most distinguishing feature is that the swap() function is expecting to receive two memory locations of integer variables where the no\_swap() function is expecting two integer values. This can be seen by the asterisks (\*) before the variables used within the swap() function. The no\_swap() function does not have the asterisks before any of the variables it uses. Listing 2 is the no\_swap() function and Listing 3 is the swap() function.

## **Listing 2**

void no\_swap(int x, int y)

{

int z; /\* Temporary local variable \*/

z = x; /\* Execute the exchange of values\*/

x = y;

y = z;

} /\* End of no\_swap function \*/

## **Listing 3**

int swap(int \*x, int \*y)

{

int z; /\* Temporary local variable \*/

z = \*x; /\* Execute the exchange of values \*/

\*x = \*y;

\*y = z;

z = \*x +\*y;

return z;

} /\* End of swap function \*/

# Testing and Verification

Testing the code for lab 1 was easy because there were no student supplied code sections to test.

The prewritten code was verified using two methods. One was to put a break point at the beginning of the while(1) loop to ensure that the board and the PIC32 were initialized correctly. The main loop was tested by adding two a break point at the end of the loop so the code could be verified through multiple while(1) loops. While doing this a viewer was used to see the values of the variables being used throughout the program and to follow them through each while(1) loop iteration.

# Questions

The difference between a #define statement to define a constant compared to declaring an initialized variable is that the #define statement is akin to a ‘word replacement’ and a variable is a chunk of memory with a value and a location. Another important difference is that a #define statement is not contained by a ‘scope’ like a declared variable is.

If the program is changed so that the line “a = c – d;” reads “d = c – d;” the program will give an error while trying to build the program. This is because the program is trying to redefine a variable that was previously assigned as a static variable. This means the program is trying to change the value of a variable that, by declaration, can’t be changed.

There are several ways to create local variables. One way to make several varables of the same type is to do it in one line. If the variables startTime, endTime, and halfTime were needed to be declared as double type variables a easy way to do that in one like is like this:

double startTime, endTime, halfTime;

This can also be accomplished in three lines like this:

double startTime;  
 double endTime;  
 double halfTime;

Another way to make variables with in C is to declare them as a constant like this:

const <type> <variable\_name>;

When making a prototyping statement the following information is needed: function return type, function name, and the corresponding type of each input variable. For example, if I wanted to prototype a function that takes two integers, divides them, and returns that number in the form of a double you would do so like this:

double divideInts( int, int);

While developing a program it may be important to know how many cycles a portion of your program needs to be able to execute a portion of the code. This can be done by setting two break points and using the stopwatch tool to measure how many cycles it takes to go from 1 break point to the other. For lab 0 this was done for the while(1) loop in the main section of the code. To complete the operations in the while(1) loop it took 73 cycles. The code, start and stop breakpoints and the output of the stopwatch tool can be seen in the figure below.

## **Figure 1**

Graphical user interface, application

Description automatically generated

# Conclusion

The lab was a success mostly because no student supplied code was needed and that the code was provided in a fully functioning condition. A limitation of this lab was that there was no student supplied code and I think a “hello world” style exercise would be helpful. This would give a student a feel for what it is like to write a little bit of code and compile it before moving on to more advanced labs.